Northeastern University
Khoury College of Computer Sciences

$k-$Nearest Neighbors & Logistic Regression

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

Wednesday | February 4, 2026

Today’s Outline

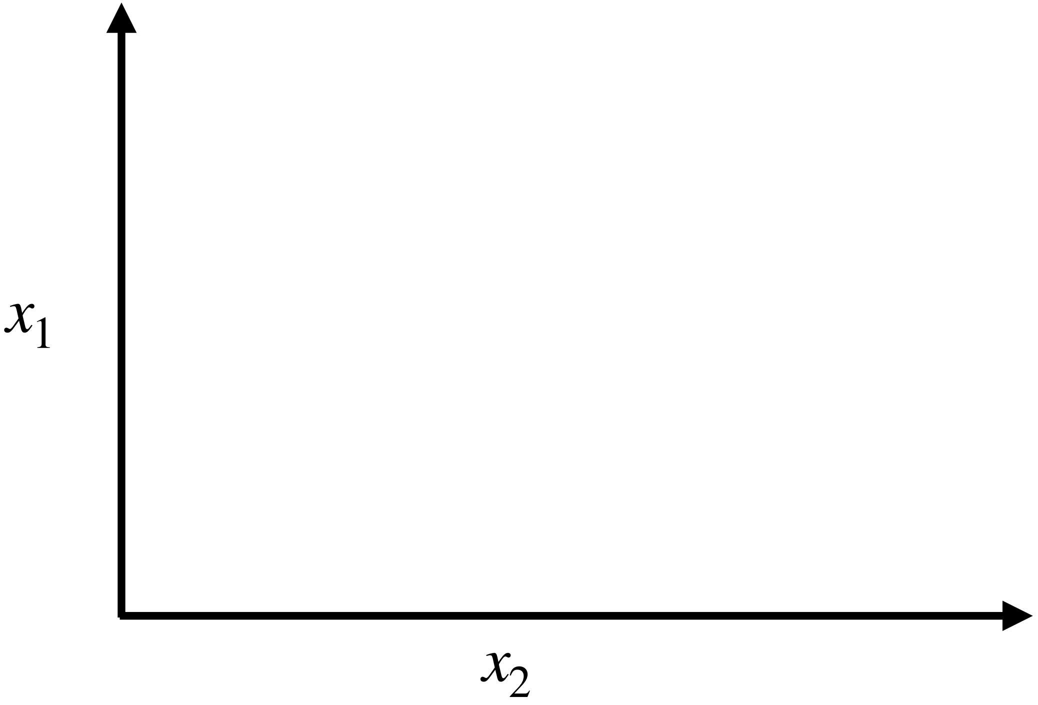
- k-Nearest Neighbors
- Logistic Regression

k-Nearest Neighbors

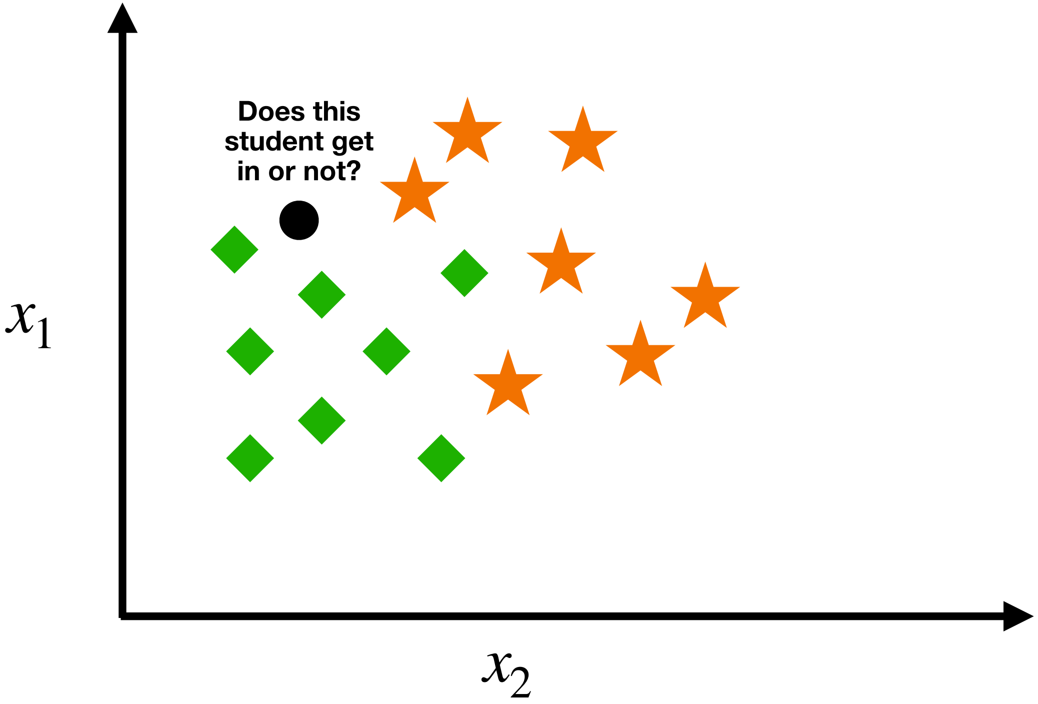
- KNN is a **non-parametric**, instance-based (lazy) learning algorithm.
- It makes no assumptions about the underlying data distribution and stores all training instances **rather than learning explicit parameters**.
- **Key Idea**:
    - Similar instances have similar labels.
    - To classify a new point, find the **K training instances closest to it** and let them vote

k-Nearest Neighbors

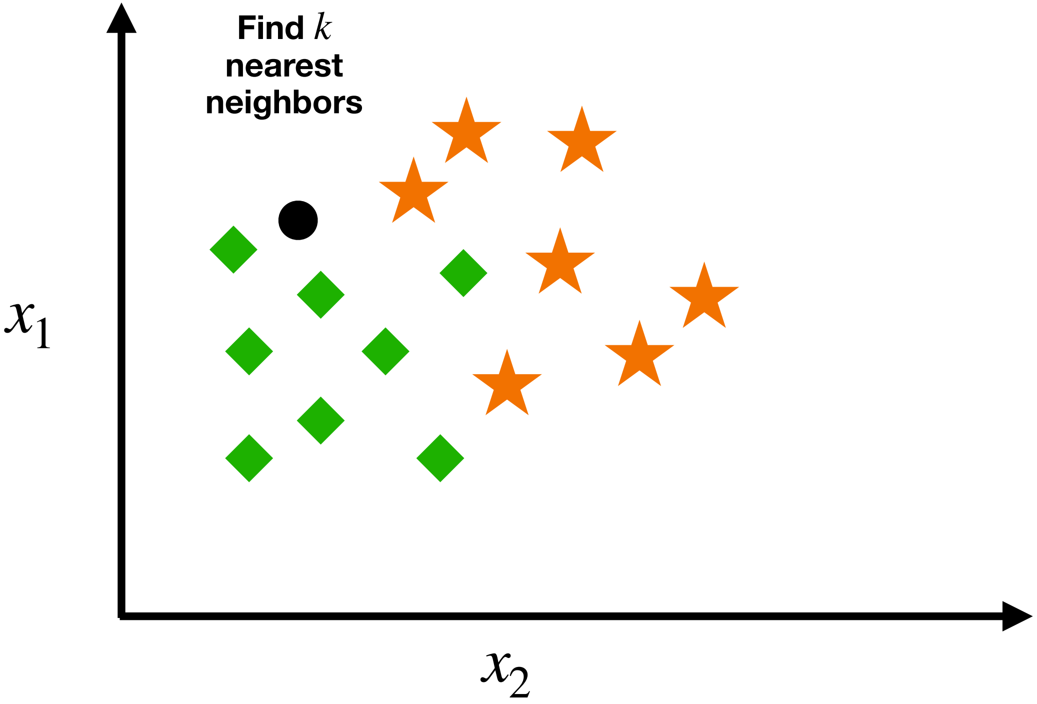
	High School GPA	SAT Scores	Get Into College?
	x_1	x_2	y
$x^{(1)}$	3.6	1500	★ = 1
$x^{(2)}$	2.7	950	◆ = 0
$x^{(3)}$	3.7	1300	★ = 1
$x^{(4)}$	3.2	1550	★ = 1
$x^{(5)}$	3.2	1000	◆ = 0
$x^{(new)}$	**3.2**	**1250**	**?**



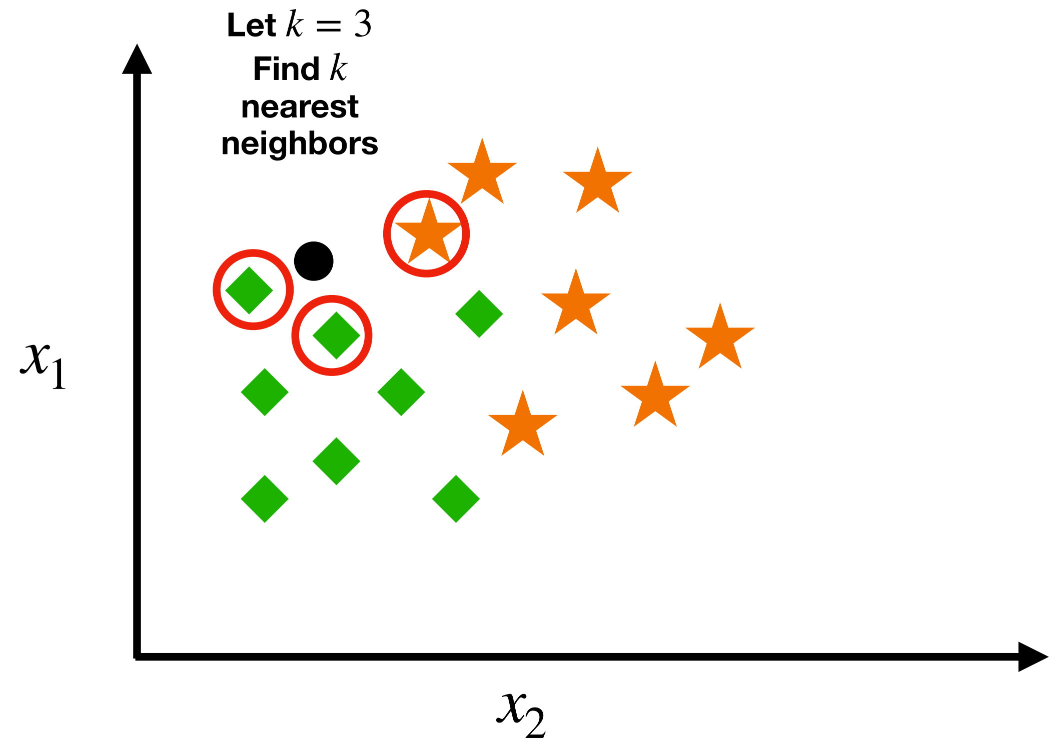
k-Nearest Neighbors



k-Nearest Neighbors



k-Nearest Neighbors

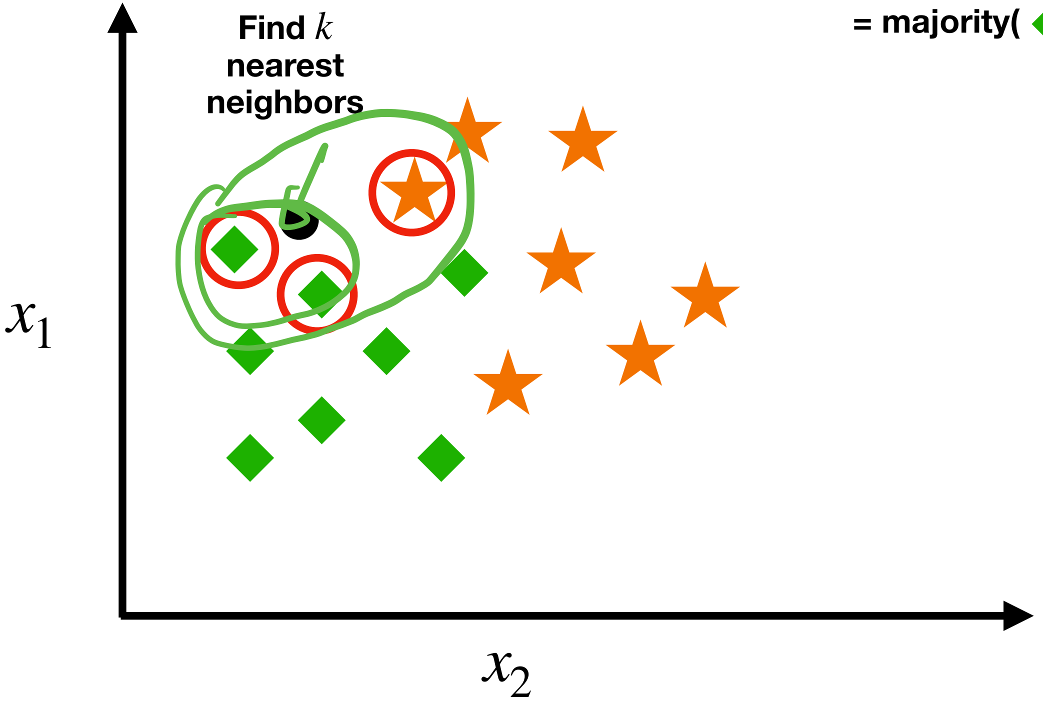


Let $k = 3$
Find k
nearest
neighbors

k-Nearest Neighbors

Let $k = 3$

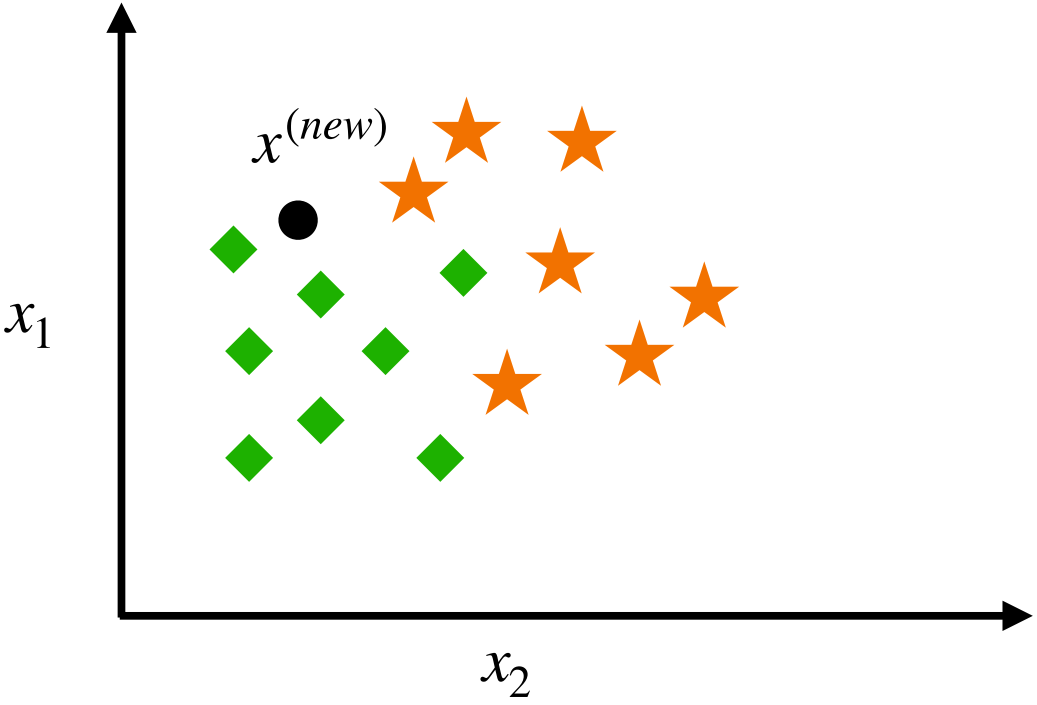
Find k nearest neighbors



Prediction for new student
= majority(◆ ◆ ★)

= ◆

k-Nearest Neighbors



Algorithm:

Training Phase:

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

Prediction/Testing Phase:

1. Compute distance from new point $x^{(new)}$ to every other point in the training data

2. Select the top $k-$nearest neighbors

3. For classification, return majority class amongst top k

4. For regression, return mean or median of the values of the $k-$neighbors

k-Nearest Neighbors

	High School GPA	SAT Scores	Get Into College?
	x_1	x_2	y
$x^{(1)}$	3.6	1500	★ = 1
$x^{(2)}$	2.7	950	◆ = 0
$x^{(3)}$	3.7	1300	★ = 1
$x^{(4)}$	3.2	1550	★ = 1
$x^{(5)}$	3.2	1000	◆ = 0
$x^{(new)}$	**3.2**	**1250**	**?**

Algorithm:

<u>Training Phase:</u>

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

<u>Prediction/Testing Phase:</u>

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

2. Select the top $k-$nearest neighbors

3. For classification, return <u>majority class</u> amongst top k

4. For regression, return <u>mean or median</u> of the values of the $k-$neighbors

k-Nearest Neighbors

The choice of the distance metric fundamentally affects which points are being considered "neighbors"

Euclidean Distance (L_2 Norm):

$$d(x^{(new)}, x^{(i)}) = \sqrt{\underbrace{\sum_{j=0}^{n} (x_j^{(new)} - x_j^{(i)})^2}_{}}$$

Manhattan Distance (L_1 Norm):

$$d(x^{(new)}, x^{(i)}) = \sum_{j=0}^{n} \underbrace{|x_j^{(new)} - x_j^{(i)}|}_{}$$

Cosine Similarity:

$$\left. \begin{array}{l} sim(x^{(new)}, x^{(i)}) = \dfrac{x^{(new)} \cdot x^{(i)}}{||x^{(new)}|| \, ||x^{(i)}||} \\ (distance = 1 - sim(x^{(new)}, x^{(i)})) \end{array} \right\}$$

Algorithm:

Training Phase:

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

Prediction/Testing Phase:

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

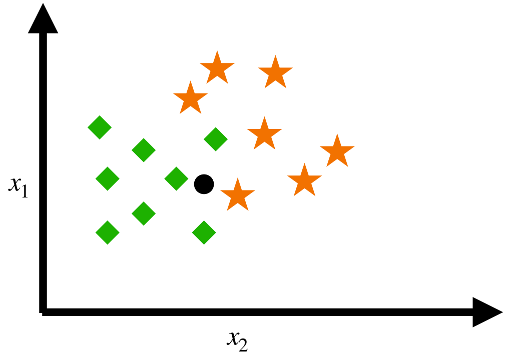
2. Select the top $k-$nearest neighbors

3. For classification, return majority class amongst top k

4. For regression, return mean or median of the values of the $k-$neighbors

k-Nearest Neighbors

Choosing k



- k is the primary hyper-parameter controlling the bias-variance tradeoff

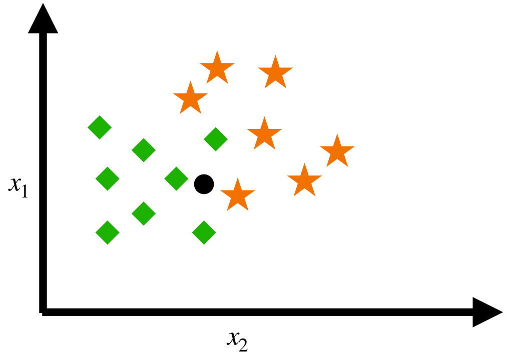
Small k (e.g. $k = 1$)

- High variance, low bias
- Decision boundary is highly irregular
- Very sensitive to noise and outliers
- Prone to overfitting, but can capture fine grained structure

Large k (e.g. $\boxed{k = m}$)

k-Nearest Neighbors

Choosing k



- k is the primary hyper-parameter controlling the bias-variance tradeoff

Small k (e.g. $k = 1$)

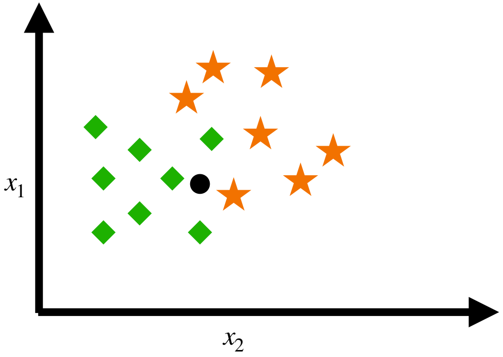
- High variance, low bias
- Decision boundary is highly irregular
- Very sensitive to noise and outliers
- Prone to overfitting, but can capture fine grained structure

Large k (e.g. $k = m$)

- High bias, low variance
- Decision boundary is very smooth
- Robust to noise, but may miss local patterns
- At the extreme of $k = m$, always predicts majority class

k-Nearest Neighbors

Choosing k



- k is the primary hyper-parameter controlling the bias-variance tradeoff

Small k (e.g. $k = 1$)

- High variance, low bias
- Decision boundary is highly irregular
- Very sensitive to noise and outliers
- Prone to overfitting, but can capture fine grained structure

Practical Tips

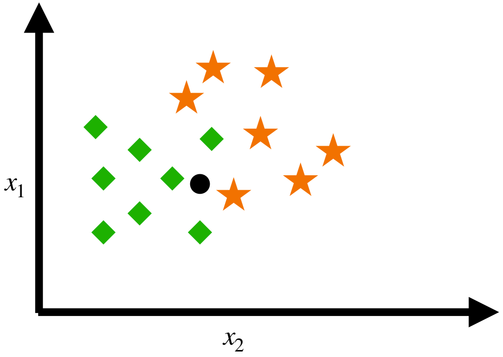
- Start with $k = \sqrt{m}$
- Use cross-validation to select optimal k
- If k is odd, it avoids ties in binary classification
- k should be smaller than the smallest class size

Large k (e.g. $k = m$)

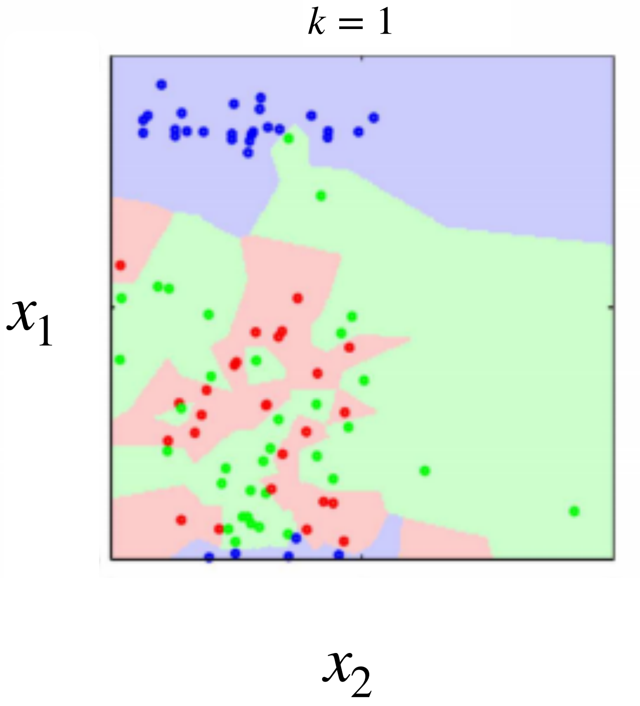
- High bias, low variance
- Decision boundary is very smooth
- Robust to noise, but may miss local patterns
- At the extreme of $k = m$, always predicts majority class

k-Nearest Neighbors

Choosing k

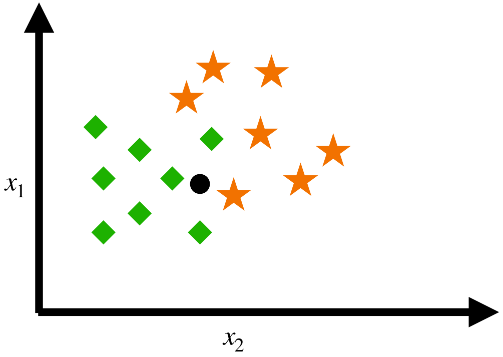


- k is the primary hyper-parameter controlling the bias-variance tradeoff

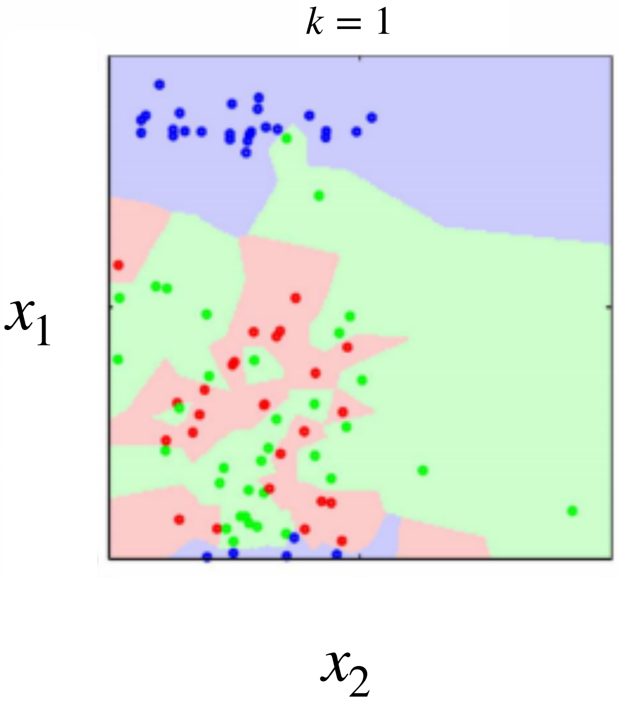


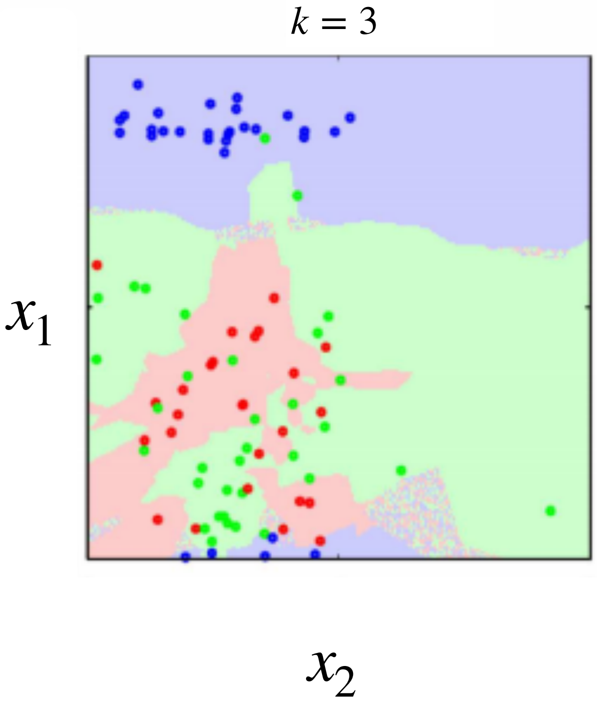
k-Nearest Neighbors

Choosing k



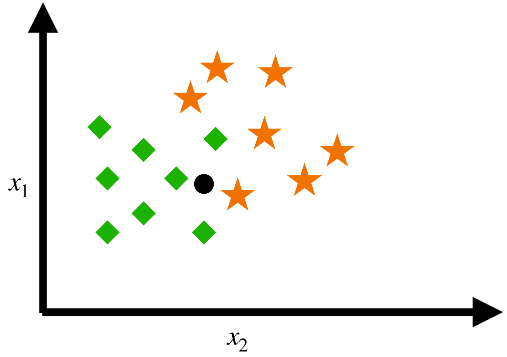
- k is the primary hyper-parameter controlling the bias-variance tradeoff



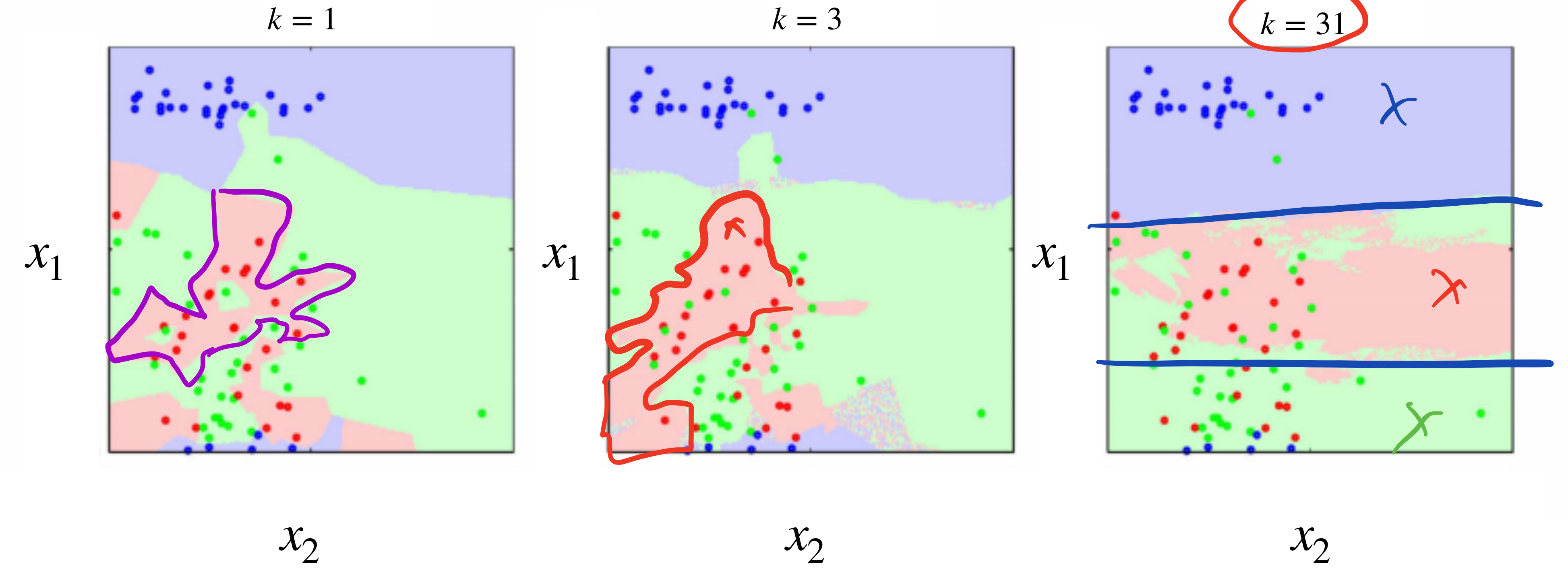


k-Nearest Neighbors

Choosing k

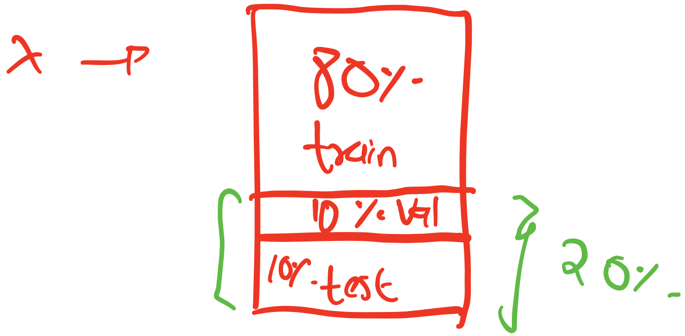


- k is the primary hyper-parameter controlling the bias-variance tradeoff



k-Nearest Neighbors

Choosing k - Cross-validation



- Why Not Just Use Training Error?
  - A model that memorizes the training data achieves zero training error but fails on new data.
  - Training error is a biased (optimistic) estimate of true generalization performance.
  - We need to estimate how well our model will perform on **unseen data**.

k-Nearest Neighbors

Choosing k - Cross-validation

- Naive Solution - Train/Test Split
    - Split data into training set (say 80%) and test set (20%).
    - Train on training set, evaluate on test set.

k-Nearest Neighbors

Choosing k - Cross-validation

- Naive Solution - Train/Test Split
  - Split data into training set (say 80%) and test set (20%).
  - Train on training set, evaluate on test set.
  - Issues:
    - **Wastes data** - 20% of precious labeled data is never used for training
    - High variance - Performance estimate depends heavily on **which points land in the test set**
    - No hyperparameter tuning: If we use the test set to select hyperparameters, we're overfitting to the test set - (using **validation set** is a possible fix for this issue)

k-Nearest Neighbors

Choosing k - Cross-validation

- Naive Solution - Train/Test Split
  - **Data Leakage Issue**
    - If we repeatedly evaluate on the test set while tuning hyperparameters, information about the test set **leaks** into our model selection process.
    - The test error becomes optimistically biased - no longer a valid estimate of **generalization**

k-Nearest Neighbors

Choosing k - Cross-validation

- **Solution!**
- Use cross-validation
  - Use **all data** for both training and validation
  - Get **reliable performance estimates** with uncertainty quantification
  - Select hyperparameters **without contaminating the final test set**

Different k from $k-$nearest neighbors

k-Fold Cross Validation Algorithm

	x_1	x_2	x_3	x_4
$x^{(1)}$				
$x^{(2)}$				
$x^{(3)}$				
$x^{(4)}$				
$x^{(5)}$				
$x^{(6)}$				
$x^{(7)}$				
$x^{(8)}$				
$x^{(9)}$				
$x^{(10)}$				

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm

	x_1	x_2	x_3	x_4
$x^{(1)}$				
$x^{(2)}$				
$x^{(3)}$				
$x^{(4)}$				
$x^{(5)}$				
$x^{(6)}$				
$x^{(7)}$				
$x^{(8)}$				
$x^{(9)}$				
$x^{(10)}$				

Let's say we want to run $k = 5$-fold cross validation

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

   3a. Use fold i as the validation set

   3b. Use the remaining $k - i$ folds as the training set

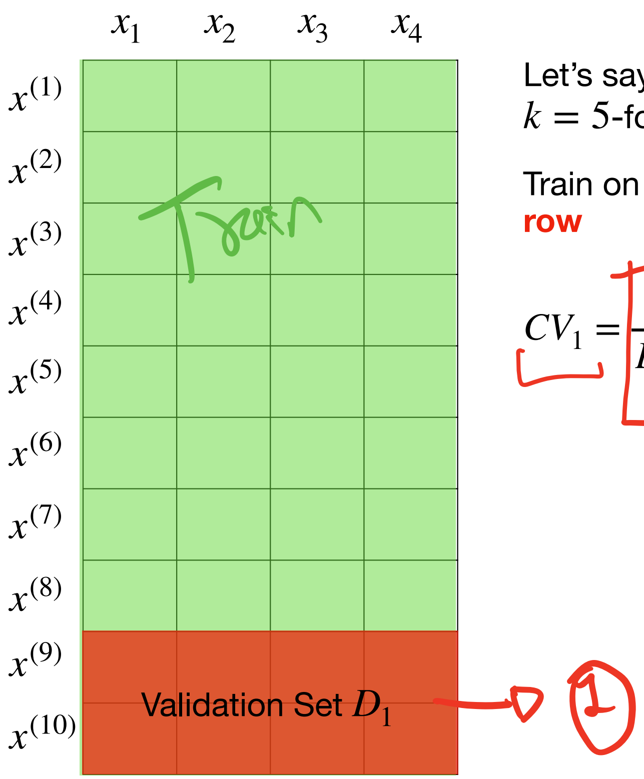
   3c. Train the model on the training set

   3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$\underbrace{CV_1}_{} = \boxed{\frac{1}{D_1} \sum_{D_1} \ell(y_{D_1}, f_\theta(D_1))}$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

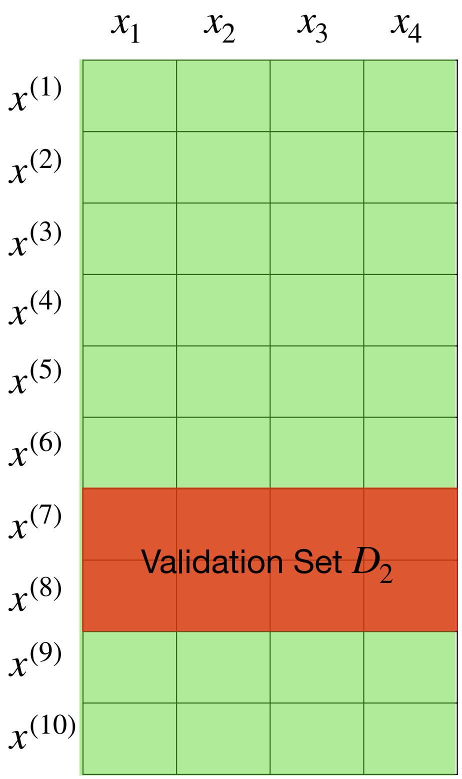
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$\boxed{CV_2 = \frac{1}{D_2} \sum_{D_2} \ell(y_{D_2}, f_\theta(D_2))}$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

   3a. Use fold i as the validation set

   3b. Use the remaining $k - i$ folds as the training set

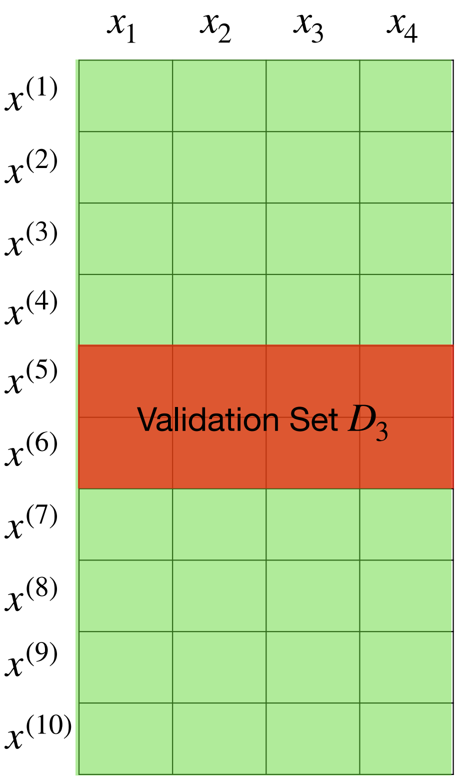
   3c. Train the model on the training set

   3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$\boxed{CV_3 = \frac{1}{D_3} \sum_{D_3} \ell(y_{D_3}, f_\theta(D_3))}$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

   3a. Use fold i as the validation set

   3b. Use the remaining $k - i$ folds as the training set

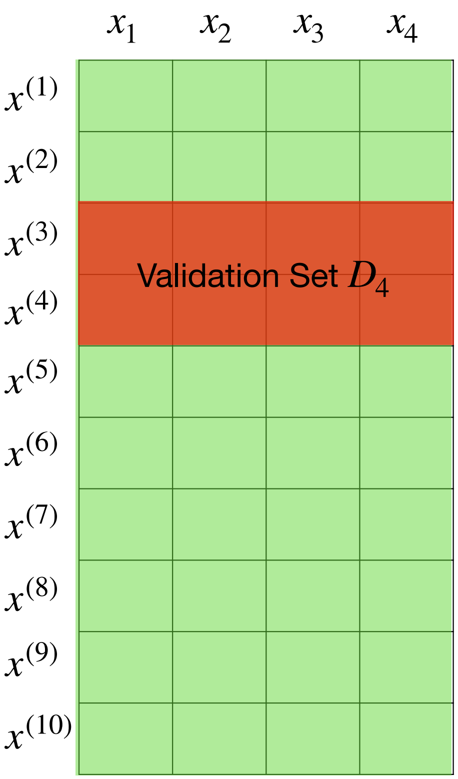
   3c. Train the model on the training set

   3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$CV_4 = \underbrace{\frac{1}{D_4}\sum_{D_4} \ell(y_{D_4}, f_\theta(D_4))}$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

   3a. Use fold i as the validation set

   3b. Use the remaining $k - i$ folds as the training set

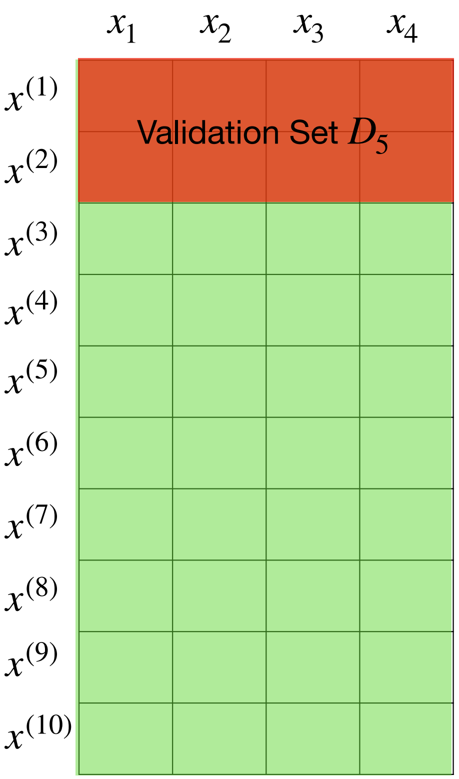
   3c. Train the model on the training set

   3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let’s say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$CV_5 = \frac{1}{D_5} \sum_{D_5} \ell(y_{D_5}, f_\theta(D_5))$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

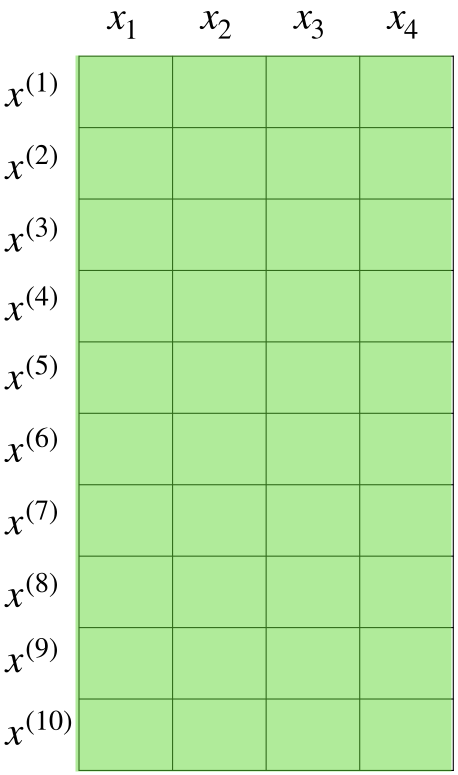
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

Mean CV Score:

$$\boxed{\bar{CV} = \frac{1}{k}\sum_{i=1}^{k} CV_i}$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

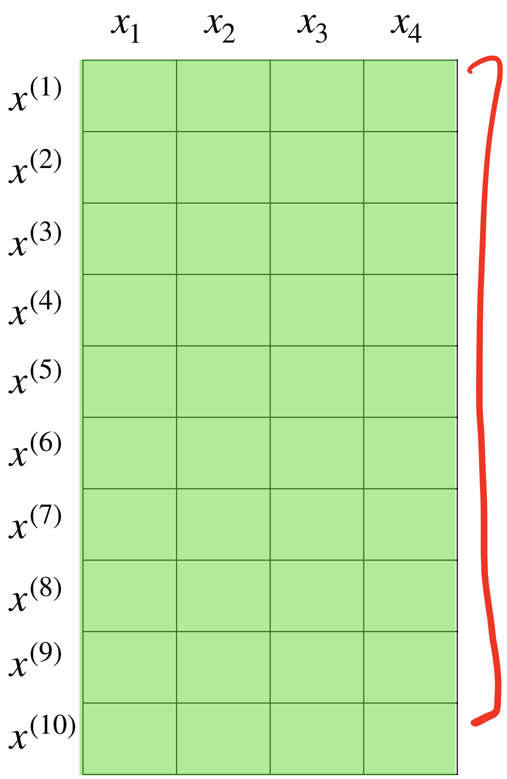
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



LOOC

Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

Mean CV Score:

$$\bar{CV} = \frac{1}{k}\sum_{i=1}^{k} CV_i$$

k-value	Training Size	Properties
k=2	50%	High Bias Low Variance Fast
k=5	80%	Good Balance Commonly Used
k=10	90%	Low Bias Commonly Used
k=m-1	m-1 samples	Low Bias Highest Variance Slow

k-Fold Cross Validation

Algorithm

	x_1	x_2	x_3	x_4
$x^{(1)}$	🟩	🟩	🟩	🟩
$x^{(2)}$	🟩	🟩	🟩	🟩
$x^{(3)}$	🟩	🟩	🟩	🟩
$x^{(4)}$	🟩	🟩	🟩	🟩
$x^{(5)}$	🟩	🟩	🟩	🟩
$x^{(6)}$	🟩	🟩	🟩	🟩
$x^{(7)}$	🟩	🟩	🟩	🟩
$x^{(8)}$	🟩	🟩	🟩	🟩
$x^{(9)}$	🟩	🟩	🟩	🟩
$x^{(10)}$	🟩	🟩	🟩	🟩

Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

Mean CV Score:

$$\bar{CV} = \frac{1}{k}\sum_{i=1}^{k} CV_i$$

k-fold CV requires **training k models.**

If training is expensive, smaller k is preferred.

k-value	**Training Size**	**Properties**
k=2	50%	High Bias Low Variance Fast
k=5	80%	Good Balance Commonly Used
k=10	90%	Low Bias Commonly Used
k=m-1	m-1 samples	Low Bias Highest Variance Slow

k-Fold Cross Validation

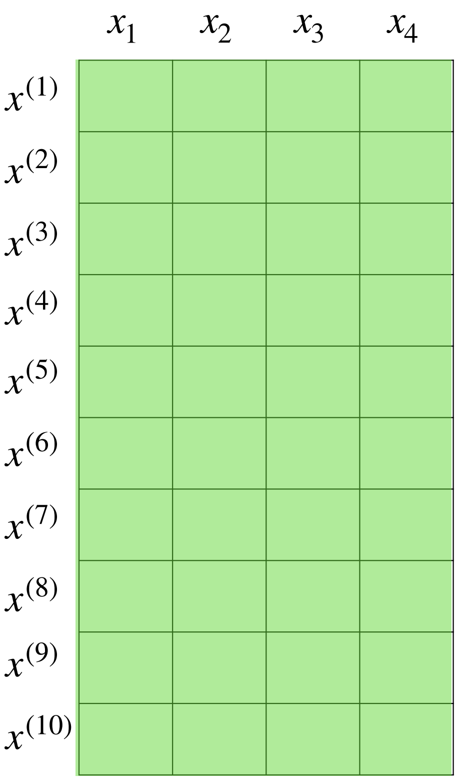
Variants

1000 — images.

900. : 100

cat dogs.

10 10



Stratified Cross-Validation

- The Problem with Random Splits
  - For imbalanced classification, random splits may create folds with different class distributions.
  - **One fold might have 40% positives while another has 20%, leading to unreliable estimates.**
  - Stratified sampling ensures each fold has approximately the same class distribution as the full dataset.
  - Algorithm:
    - Separate samples by class
    - For each class, distribute samples evenly across $k-$folds
    - Combine to form final folds

Back to k-Nearest Neighbors

Practical Issues

- GPA → 0 - 430

- SAT → 0 - 1600

- Feature Scaling

- Curse of Dimensionality

- Space and computational complexity

$$\text{distance} = \sqrt{\sum_{i=0}^{2} \left(x_i^{(1)} - x_i^{(2)} \right)^2}$$

Back to k-Nearest Neighbors

Practical Issues - Feature Scaling

- KNN is **highly sensitive** to feature scales because distance metrics are dominated by features with larger ranges.
- Example:
  - If feature A ranges from 0-1 and feature B ranges from 0-1000
  - Euclidean distance is almost **entirely determined by feature B**.
  - Solution: Always normalize or standardize features before applying kNN.

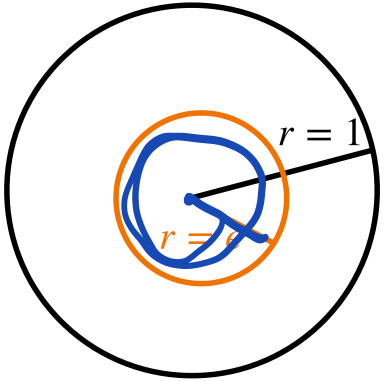
GPA SAT

Back to k-Nearest Neighbors

Practical Issues - Curse of Dimensionality

- KNN suffers severely in high-dimensional spaces:
  - Distance concentration: As dimensionality increases, **distances between points become increasingly similar**.
  - **The ratio of nearest to farthest neighbor approaches 1**, making the concept of "nearest" meaningless.

$$\frac{\cancel{\pi}(1-\epsilon)^2}{\cancel{\pi}\,1^2} = \frac{(1-\epsilon)^d}{1^d} \to$$



Back to k-Nearest Neighbors

Practical Issues - Curse of Dimensionality

- KNN suffers severely in high-dimensional spaces:
  - Distance concentration: As dimensionality increases, **distances between points become increasingly similar**.
  - **The ratio of nearest to farthest neighbor approaches 1**, making the concept of “nearest” meaningless.
- Irrelevant features: In high dimensions, many features may be irrelevant, adding noise to distance calculations.
- Mitigation strategies:
  - Dimensionality reduction (PCA, feature selection)
  - **Feature weighting** based on relevance
  - Consider other algorithms for d > 20

GPA $\boxed{\text{SAT}}$ . . – .

Back to k-Nearest Neighbors

Practical Issues - Computational Complexity

- Training: $O(1)$ - just store the data

Back to k-Nearest Neighbors

Practical Issues - Computational Complexity

- Training: $O(1)$ - just store the data
- Prediction (naive):
    - $\boxed{O(nm)}$ per query, where m is training set size and n is dimensionality.
    - Must compute distance to **all** m points.

Back to k-Nearest Neighbors

Practical Issues - Computational Complexity

- Training: $O(1)$ - just store the data
- Prediction (naive):
    - $O(nm)$ per query, where m is training set size and n is dimensionality.
    - Must compute distance to **all** m points.
- Prediction (optimized) - Data structures can accelerate nearest neighbor search:
    - **KD-trees**: $O(\underline{nlogm})$ average case for low dimensions, but degrades to $O(nm)$ in high dimensions
    - **Ball trees**: Better for high dimensions than KD-trees
    - **Locality-sensitive hashing (LSH)**: Approximate nearest neighbors in $O(n)$ with preprocessing

Back to k-Nearest Neighbors

Practical Issues - Computational Complexity

$$y = \underset{1}{\theta_0} + \underset{1}{\theta_1} x$$

- Training: $O(1)$ - just store the data
- Prediction (naive):
    - $O(nm)$ per query, where m is training set size and n is dimensionality.
    - Must compute distance to **all** m points.
- Prediction (optimized) - Data structures can accelerate nearest neighbor search:
    - **KD-trees**: $O(nlogm)$ average case for low dimensions, but degrades to $O(nm)$ in high dimensions
    - **Ball trees**: Better for high dimensions than KD-trees
    - **Locality-sensitive hashing (LSH)**: Approximate nearest neighbors in $O(n)$ with preprocessing
- Space complexity: $\underbrace{O(nm)}$ to store the training data.

$$\propto \frac{(1-\varepsilon)^d}{1^d}$$

Back to k-Nearest Neighbors

Practical Issues

$$y = \underbrace{\theta_0 x_1}_{SAT} + \underbrace{\theta_1 x_2}_{GPA} + \underbrace{\theta_2}_{0} \boxed{x_3}$$

Pros

- Simple to understand and implement
- No training phase (fast to “train”)
- Naturally handles multi-class classification
- Non-parametric: makes no distributional assumptions
- Can capture arbitrarily complex decision boundaries
- Easily adapts to new training data (just add it)

Cons

- Slow prediction for large datasets
- High memory requirement (stores all training data)
- Sensitive to irrelevant features and feature scaling
- Struggles in high dimensions (curse of dimensionality)
- No interpretable model or feature importance
- Requires meaningful distance metric

Back to k-Nearest Neighbors

When to use k-NN?

Use

- Small to medium datasets
- Low to moderate dimensionality ($n < 20$)
- Non-linear decision boundaries expected
- Data arrives incrementally (online learning)
- Quick baseline model needed

Don't Use

- Large datasets with real-time prediction requirements
- Very high-dimensional data
- Features have varying relevance
- Interpretability is required

Today’s Outline

- k-Nearest Neighbors
- **Logistic Regression**

Logistic Regression

① Model: $\hat{y} = \theta_0 + \theta_1 x$.

② Loss : Mean Squared Error : $\frac{1}{m}(\overset{\text{true}}{y} - \overset{\text{predicted.}}{\hat{y}})^2 \}$

③ Differentiate Loss → set 0 → closed form

→ Gradient descent.

① Model.

②

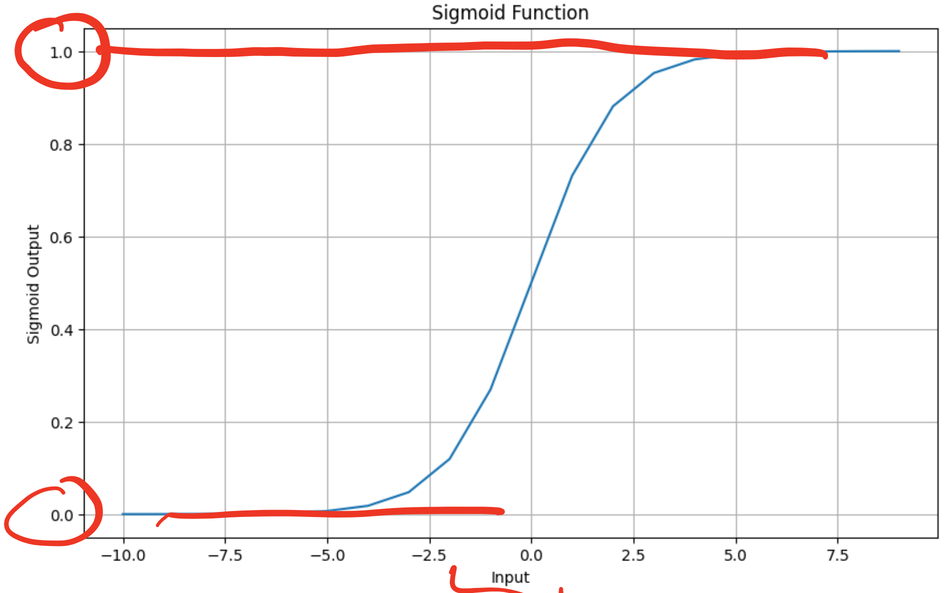
③

Logistic Regression

- Despite its name, logistic regression is a **classification** algorithm.
- It models the probability of class membership using a logistic (sigmoid) function.

Sigmoid. —
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

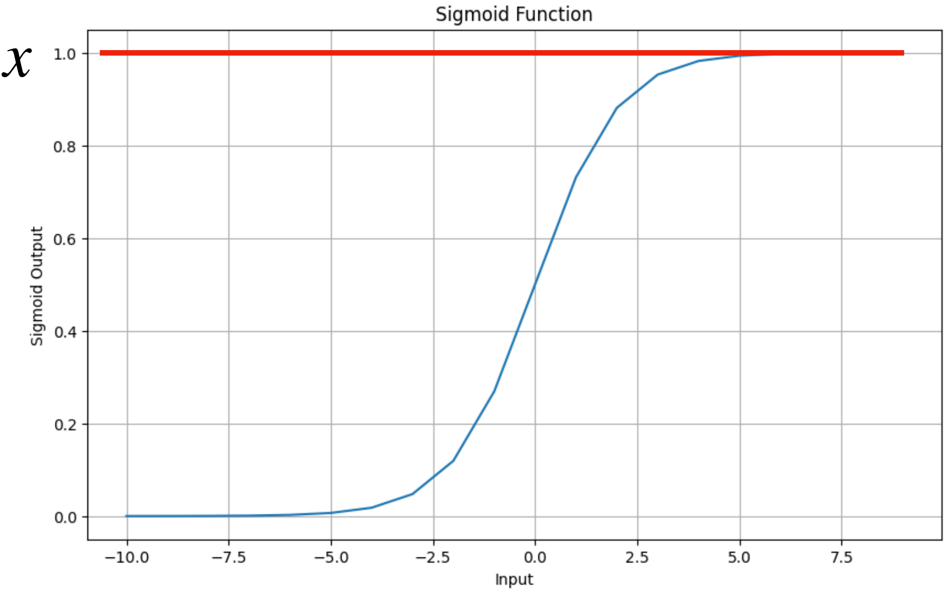
$$\sigma(\theta_0 + \theta_1 x) = \hat{y}$$



Logistic Regression

- Despite its name, logistic regression is a **classification** algorithm.
- It models the probability of class membership using a logistic (sigmoid) function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

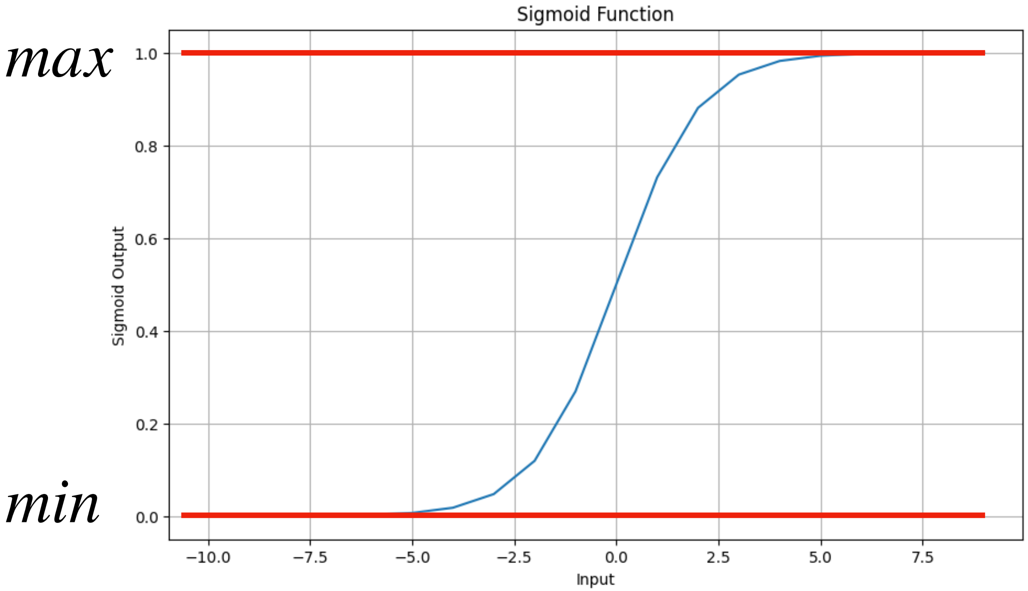


max

Logistic Regression

- Despite its name, logistic regression is a **classification** algorithm.
- It models the probability of class membership using a logistic (sigmoid) function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

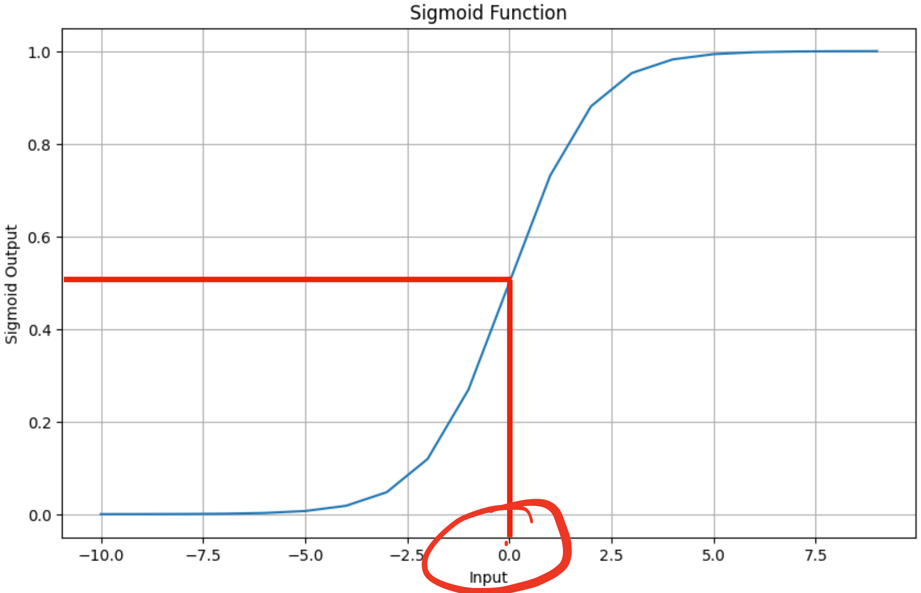


Logistic Regression

- Despite its name, logistic regression is a **classification** algorithm.
- It models the probability of class membership using a logistic (sigmoid) function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$mid = 0.5$$

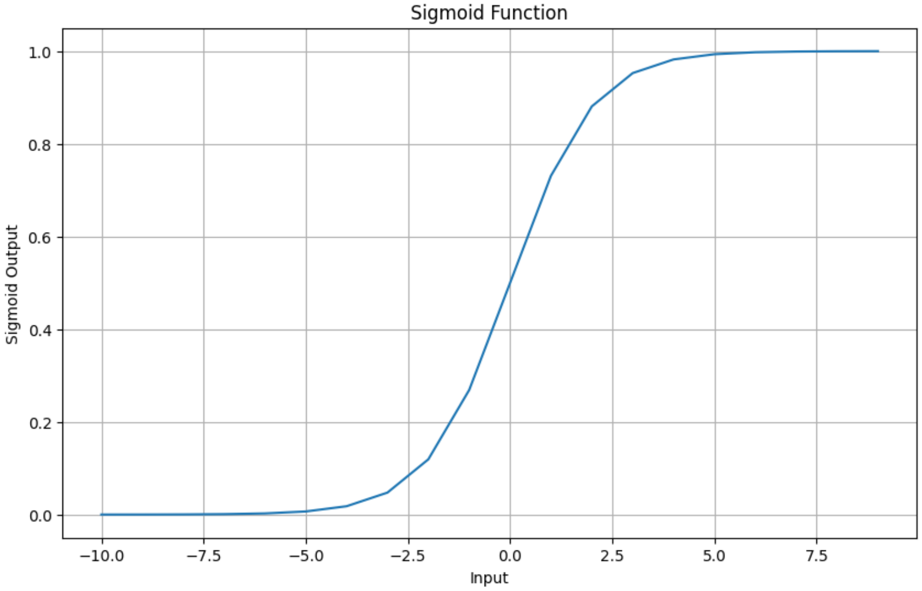


Logistic Regression

- Despite its name, logistic <u>regression</u> is a **classification** algorithm.
- It models the probability of class membership using a logistic (sigmoid) function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Linear regression predicts **unbounded** real values as $\underset{\sigma}{\hat{y} =} (\theta_0 + \theta_1 \cdot x)$
- But we need probabilities in $[0,\ 1]$



Logistic Regression

$\theta_0 + \theta_1 x < 0$

- Wrap the linear regression equation in a Sigmoid function
- Logistics regression models the probability of the positive class

$$\mathbb{P}(Y = 1 \,|\, X = x) = \boxed{\sigma(\theta_0 + \theta_1 \cdot x)} = \frac{1}{1 + e^{-(\theta_0 + \theta_1 \cdot x)}}$$

The decision boundary is the hyperplane where $\mathbb{P}(Y = 1 \,|\, X = x) = \underline{0.5}$, which occurs when $\underline{\theta_0 + \theta_1 \cdot x = 0}$

$\sigma(0) = 50\%.$

$\sigma(\cdot > 0) = 1$

$\sigma(\cdot < 0) - -1$

Logistic Regression

The decision boundary is the hyperplane where $\mathbb{P}(Y = 1 \,|\, X = x) = 0.5$, which occurs when $\theta_0 + \theta_1 \cdot x = 0$

Assume that threshold = 0.5

If $\theta_0 + \theta_1 \cdot x \geq 0$, classify as "positive class"
Why?

Logistic Regression

The decision boundary is the hyperplane where $\mathbb{P}(Y = 1 \mid X = x) = 0.5$, which occurs when $\theta_0 + \theta_1 \cdot x = 0$

Assume that threshold = 0.5

If $\boxed{\theta_0 + \theta_1 \cdot x \geq 0}$ classify as “positive class”

Why?

Because $\sigma(k \geq 0) \geq 0.5$

Logistic Regression

The decision boundary is the hyperplane where $\mathbb{P}(Y = 1 \mid X = x) = 0.5$, which occurs when $\theta_0 + \theta_1 \cdot x = 0$

Assume that threshold = 0.5

If $\theta_0 + \theta_1 \cdot x \geq 0$, classify as “positive class”

Why?

Because $\sigma(k \geq 0) \geq 0.5$

If $\boxed{\theta_0 + \theta_1 \cdot x \leq 0}$, classify as “negative class”

Why?

Logistic Regression

Model: $\hat{y} = \sigma\left(\boxed{\theta_0+\theta_1 x}\right)$

$z = 0$

$\sigma(z) = 0.5$

$z > 0$

$\sigma(z) > 0.5$ +ve

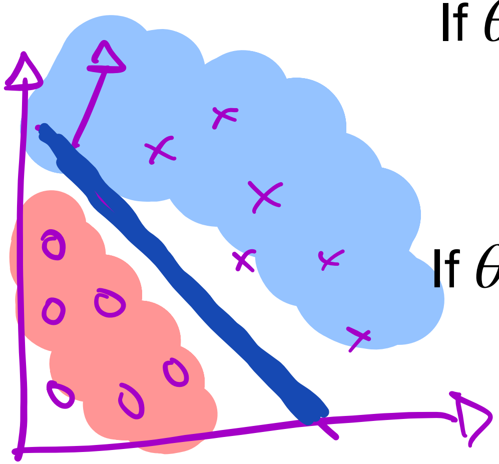
$z < 0$

$\sigma(z) < 0.5$ -ve

The decision boundary is the hyperplane where $\mathbb{P}(Y = 1 \,|\, X = x) = 0.5$, which occurs when $\theta_0 + \theta_1 \cdot x = 0$

$\theta_0+\theta_1 x^{(new)}$

Assume that threshold = 0.5



If $\theta_0 + \theta_1 \cdot x \geq 0$, classify as "positive class"

Why?

Because $\sigma(k \geq 0) \geq 0.5$

If $\theta_0 + \theta_1 \cdot x \leq 0$, classify as "negative class"

Why?

Because $\sigma(k \leq 0) \leq 0.5$

Logistic Regression

Model:

$$\boxed{\hat{y} = \sigma(\theta_0 + \theta_1 \cdot x)}$$

Loss:

$$\boxed{\ell(\theta) = \frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(\hat{y^{(i)}}) + (1 - y^{(i)}) log(1 - \hat{y^{(i)}})}$$

Logistic Regression

$\boxed{\theta}$

How do we train this?

Maximum Likelihood Estimation

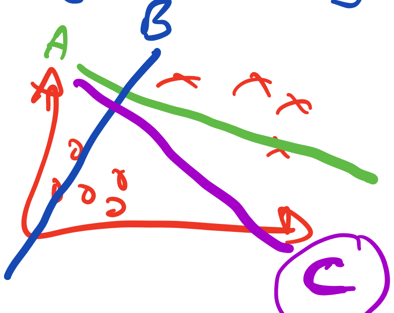
Maximum Likelihood Estimation (MLE) is a principled method for **estimating the parameters of a statistical model**.

Key Idea - Choose parameters that make the observed data **most probable**.

Logistic Regression

How do we train this? $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$

$$D = \{ (\underset{\text{input}}{x^{(1)}}, \underset{\text{output}}{y^{(1)}}), (x^{(2)}, y^{(2)}) \cdots \}$$



Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is a principled method for **estimating the parameters of a statistical model**.

Key Idea - Choose parameters that make the observed data **most probable.**

$\theta_A = \begin{bmatrix} \theta_{A0} \\ \theta_{A1} \end{bmatrix}$ — Given some dataset D and a model with parameters θ

$\theta_B = \begin{bmatrix} \theta_{B0} \\ \theta_{B1} \end{bmatrix}$ —

$$\hat{\theta}_{MLE} = \underbrace{arg\ max_\theta}_{} \boxed{\mathbb{P}(D|\theta)}$$

$\theta_C = \begin{bmatrix} \quad \end{bmatrix}$ —

Logistic Regression

How do we train this?

Maximum Likelihood Estimation

Key Idea - Choose parameters that make the observed data most probable.

Given some dataset D and a model with parameters θ

$$\hat{\theta}_{MLE} = arg\ max_{\theta} \mathbb{P}(D \mid \theta)$$

Probability that we observe training dataset D, given that the model has parameters θ

Logistic Regression

How do we train this?

Maximum Likelihood Estimation

Key Idea - Choose parameters that make the observed data most probable.

Given some dataset D and a model with parameters θ

$$\hat{\theta}_{MLE} = arg\ max_{\theta} \mathbb{P}(D \mid \theta)$$

Find θ such that this probability is maximized

Logistic Regression

How do we train this?

Maximum Likelihood Estimation

Key Idea - Choose parameters that make the observed data most probable.

Given some dataset D and a model with parameters θ

$$\hat{\theta}_{MLE} = arg\ max_{\theta}\mathbb{P}(D\,|\,\theta)$$

Under what parameter values would we have been **most likely to observe exactly the data we did observe**?

Logistic Regression

How do we train this?

Maximum Likelihood Estimation

What we want to find:

$$\hat{\theta}_{MLE} = arg\ max_{\theta} \boxed{\mathbb{P}(D\,|\,\theta)}$$

θ_A

θ_B

θ_C

Probability:

$\mathbb{P}(D\,|\,\theta)$ - Given **fixed parameters θ,**
what is the probability of observing data D?
This is a function of D with θ fixed.

Logistic Regression

How do we train this?

$\boxed{D}$

Maximum Likelihood Estimation

What we want to find:

$$\hat{\theta}_{MLE} = arg\ max_{\theta}\mathbb{P}(D\,|\,\theta)$$

Probability:

$\mathbb{P}(D\,|\,\theta)$ - Given **fixed parameters θ,**
what is the probability of observing data D?
This is a function of D with θ fixed.

Likelihood:

$\boxed{L(\theta\,|\,D) = \mathbb{P}(D\,|\,\theta)}$ - Given fixed observed data D, how **likely** are different parameter values θ?
This is a function of θ with D fixed.

Logistic Regression

Likelihood Function

$D \rightarrow \{(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})\}$

For **independent** observations (rows of data) $D = \underbrace{\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}}$,
the likelihood is the **product** of individual probabilities

$$L(\theta \,|\, D) = \underbrace{\mathbb{P}(D \,|\, \theta)} = \boxed{\Pi_{i=1}^{m}} \mathbb{P}(x^{(i)} \,|\, \theta)$$

$P(x^{(1)} | \theta) \cdot P(x^{(2)} | \theta) \cdots P(x^{(m)} | \theta)$

Logistic Regression

Likelihood Function

$a \cdot b$

$$\boxed{\log(a \cdot b) = \log a + \log b}$$

$\mathbb{P}(D|\theta)$

For **independent** observations (rows of data) $D = \{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$, the likelihood is the **product** of individual probabilities

$$\log\left(\underline{L(\theta\,|\,D) \boxed{=} \mathbb{P}(D\,|\,\theta)}\right) = \boxed{\Pi_{i=1}^{m}}\,\mathbb{P}(x^{(i)}\,|\boxed{\theta})$$

But, products are numerically **unstable** and difficult to differentiate

So, we take log on both sides to convert products to sums

$$\log(L(\theta|D)) = \underbrace{\sum_{i=1}^{m}} \log(P(x^{(i)}|\theta))$$

Logistic Regression

Likelihood Function

For **independent** observations (rows of data) $D = \{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$, the likelihood is the **product** of individual probabilities

$$L(\theta \,|\, D) = \mathbb{P}(D \,|\, \theta) = \boxed{\Pi_{i=1}^{m}} \, \mathbb{P}(x^{(i)} \,|\, \theta)$$

$$\boxed{log}(L(\theta \,|\, D)) = \boxed{\sum_{i=1}^{m}} \boxed{log}(\mathbb{P}(x^{(i)}) \,|\, \theta)$$

Using properties of log:

$$log(a^b) = b \cdot log(a)$$

$$\boxed{log(ab) = log(a) + log(b))}$$

Logistic Regression

Likelihood Function

For **independent** observations (rows of data) $D = \{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$, the likelihood is the **product** of individual probabilities

$$L(\theta \,|\, D) = \mathbb{P}(D \,|\, \theta) = \Pi_{i=1}^{m} \ \mathbb{P}(x^{(i)} \,|\, \theta)$$

$$log(L(\theta \,|\, D)) = \sum_{i=1}^{m} log(\mathbb{P}(x^{(i)}) \,|\, \theta)$$

For logistic regression

Input Features: $x \in \mathbb{R}^m$

Binary Labels: $y \in \{0,1\}$

Training Data: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(m)}, y^{(m)})\}$

Logistic Regression

① Model: $\underline{\hat{y}} = \underline{\sigma(\theta_0 + \theta_1 x)} \Rightarrow \boxed{\mathbb{P}(y=1 \mid x, \theta)}$ — $\boxed{p}$ (success, +ve, 1)

$\boxed{1-p}$ (failure, -ve, 0, -1)

② Bernoulli Distributions → $P(k;p) = \boxed{p^k \cdot (1-p)^{1-k}}$

↪ $p\,(1-p)^{1-1} = \boxed{p}$

$p^0 \cdot (1-p)^{1-0} = \boxed{1-p}$

$$L(\theta) = \mathbb{P}(D \mid \theta) = \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)}; \theta)$$

$$\log(L\theta) = \sum_{i=1}^{m} \log\left(\underline{\mathbb{P}(y^{(i)} \mid x^{(i)}; \theta)}\right)$$

Logistic Regression

$$\log(L\theta) = \sum_{i=1}^{m} \log\left(\underbrace{P(y^{(i)} \mid x^{(i)}, \theta)}\right) - \quad \sim(\ )$$

$\log(ab) = \log a + \log b.$

$\boxed{\log(a^{b}) = b\log a}$

$$\log(L(\theta)) = \sum_{i=1}^{m} \log\left(p_i^{y_i} \cdot (1-p_i)^{(1-y_i)}\right)$$

true. predicted

$$\log(L(\theta)) = \sum_{i=1}^{m} \overbrace{y_i} \log \overbrace{p_i} + (1-y_i)\log(1-p_i)$$

Log-likelihood function.

$y_i = 1 \quad = \quad 1\cdot\log\underbrace{p_i} \quad -\infty$

$1\cdot 0 \quad -0$

$y_i = 0 \quad 1\cdot\log(\underbrace{1-p_i}_{0}) = \infty$

Logistic Regression

$$\sigma(\theta_0+\theta_1 x^{(i)}) = p^{(i)}$$

$$-\log(\mathcal{L}(\theta)) = \sum_{i=1}^{m} y_i \log p_i + (1-y_i)\log(1-p_i)$$

maximize

$$\operatorname*{argmax}_{\theta} \mathbb{P}(D|\theta)$$

$$\text{maximize } f = \text{minimize } -f$$

$$\rightarrow -\log(\mathcal{L}(\theta))$$

$$\hat{y} = \sigma(\theta_0+\theta_1 x)$$

$$p = \begin{cases} p & \text{if } y=1 \\ 1-p & \text{if } y=0 \end{cases} \rightarrow p^{y} \cdot (1-p)^{(1-y)}$$

$$y = \{0, 1\}$$

Logistic Regression

$$\mathbb{P}(Y = 1 \mid X = x; \theta) = \sigma(\theta_0 + \theta_1 \cdot x)$$

Logistic Regression

$$\mathbb{P}(Y = 1 \mid X = x; \theta) = \sigma(\theta_0 + \theta_1 \cdot x)$$

Each label y_i follows a **Bernoulli Distribution** with parameter

$$p_i = \mathbb{P}(Y = 1 \mid x_i)$$

Logistic Regression

Quick Aside: Bernoulli Distribution

Bernoulli Distribution models a single binary outcome

Is $\mathbb{P}(X = success) = p$ and
$\mathbb{P}(X = failure) = q = (1 - p)$

Then probability mass function P is

$$P(X = x) = p^x \cdot (1 - p)^{1-x}$$

Logistic Regression

$$\mathbb{P}(Y = 1 \mid X = x; \theta) = \sigma(\theta_0 + \theta_1 \cdot x)$$

Each label y_i follows a **Bernoulli Distribution** with parameter

$$p_i = \mathbb{P}(Y = 1 \mid x_i)$$

$$\mathbb{P}(Y = y \mid X = x) = p^y (1 - p)^{1-y}$$

Logistic Regression

$$\mathbb{P}(Y = 1 \mid X = x; \theta) = \sigma(\theta_0 + \theta_1 \cdot x)$$

Each label y_i follows a **Bernoulli Distribution** with parameter

$$p_i = \mathbb{P}(Y = 1 \mid x_i)$$

$$\mathbb{P}(Y = y \mid X = x) = p^y (1 - p)^{1-y}$$

When $y = 1 \rightarrow p^1 (1 - p)^0 = p$

When $y = 0 \rightarrow p^0 (1 - p)^1 = (1 - p)$

Logistic Regression

$$\mathbb{P}(Y = 1 \,|\, X = x; \theta) = \sigma(\theta_0 + \theta_1 \cdot x)$$

Each label y_i follows a **Bernoulli Distribution** with parameter

$$p_i = \mathbb{P}(Y = 1 \,|\, x_i)$$

$$\mathbb{P}(Y = y \,|\, X = x) = p^y(1 - p)^{1-y}$$

When $y = 1 \rightarrow p^1(1 - p)^0 = p$

When $y = 0 \rightarrow p^0(1 - p)^1 = (1 - p)$

Logistic Regression

For a **single** observation $(x^{(i)}, y^{(i)})$

Probability of observing $y^{(i)}$ given you have seen input data $x^{(i)}$ and θ

$$\mathbb{P}(y^{(i)} \,|\, x^{(i)}; \theta) = p_i^{y^{(i)}}(1 - p_i)^{1-y^{(i)}}$$

Where $p_i = \sigma(\theta_0 + \theta_1 \cdot x)$

Logistic Regression

For the **entire dataset** $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots (x^{(m)}, y^{(m)})\}$

Assuming observations are **independent**

Likelihood is the product of all individual probabilities

$$L(\theta \,|\, D) = \Pi_{i=1}^{m} \mathbb{P}(y^{(i)} \,|\, x^{(i)}; \theta) = \Pi_{i=1}^{m} p_i^{y^{(i)}} (1 - p_i)^{1 - y^{(i)}}$$

Logistic Regression

For the **entire dataset** $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots (x^{(m)}, y^{(m)})\}$

Assuming observations are **independent**

Likelihood is the product of all individual probabilities

$$L(\theta | D) = \Pi_{i=1}^{m} \mathbb{P}(y^{(i)} | x^{(i)}; \theta) = \Pi_{i=1}^{m} p_i^{y^{(i)}} (1 - p_i)^{1-y^{(i)}}$$

We want to **maximize** likelihood

$$\hat{\theta}_{MLE} = arg\ max_{\theta} \mathbb{P}(D | \theta)$$

Logistic Regression

$$L(\theta|D) = \Pi_{i=1}^{m} p_i^{y^{(i)}} (1 - p_i)^{1-y^{(i)}}$$

Logistic Regression

$$L(\theta|D) = \Pi_{i=1}^{m} p_i^{y^{(i)}}(1 - p_i)^{1-y^{(i)}}$$

$$log(L(\theta)) = log(\Pi_{i=1}^{m} p_i^{y^{(i)}}(1 - p_i)^{1-y^{(i)}})$$

Using properties of log:

$$log(a^b) = b \cdot log(a)$$
$$log(ab) = log(a) + log(b))$$

$$log(L(\theta)) = \sum_{i=1}^{m} log(p_i^{y^{(i)}}(1 - p_i)^{1-y^{(i)}})$$

$$log(L(\theta)) = \sum_{i=1}^{m} y^{(i)}log(p_i) + (1 - y^{(i)})log(1 - p_i)$$

Logistic Regression

$$L(\theta | D) = \Pi_{i=1}^{m} p_i^{y^{(i)}}(1 - p_i)^{1-y^{(i)}}$$

$$log(L(\theta)) = log(\Pi_{i=1}^{m} p_i^{y^{(i)}}(1 - p_i)^{1-y^{(i)}})$$

Using properties of log:

$$log(a^b) = b \cdot log(a)$$
$$log(ab) = log(a) + log(b))$$

$$log(L(\theta)) = \sum_{i=1}^{m} log(p_i^{y^{(i)}}(1 - p_i)^{1-y^{(i)}})$$

$$log(L(\theta)) = \sum_{i=1}^{m} y^{(i)} log(p_i) + (1 - y^{(i)}) log(1 - p_i)$$

This is called the **log-likelihood** function for logistic regression

Logistic Regression

$$log(L(\theta)) = \sum_{i=1}^{m} y^{(i)} log(p_i) + (1 - y^{(i)}) log(1 - p_i)$$

This is called the **log-likelihood** function for logistic regression

Remember we want to **maximize** likelihood

But when we deal with “loss” functions and gradient descent, we want to **minimize** the loss

Logistic Regression

$$\ell(\theta) = -\sum_{i=1}^{m} y^{(i)} log(p_i) + (1 - y^{(i)}) log(1 - p_i)$$

Solution: Minimize **negative** likelihood

Logistic Regression

$$\ell(\theta) = -\sum_{i=1}^{m} y^{(i)} log(p_i) + (1 - y^{(i)}) log(1 - p_i)$$

Solution: Minimize **negative** likelihood

Remember that p_i is the predicted output where

$$p_i = \sigma(\theta_0 + \theta_1 \cdot x)$$

Logistic Regression

$$\ell(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}log(\hat{y}^{(i)}) + (1 - y^{(i)})log(1 - \hat{y}^{(i)})$$

Binary Cross Entropy Loss

Logistic Regression

$$\ell(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}log(\hat{y}^{(i)}) + (1 - y^{(i)})log(1 - \hat{y}^{(i)})$$

When $y^{(i)} = 1$, i.e., actual positive

$$\ell(\theta) = -\,log(\hat{y}^{(i)})$$

When $y^{(i)} = 0$, i.e., actual negative

$$\ell(\theta) = -\,log(1 - \hat{y}^{(i)})$$

Logistic Regression

$$\ell(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)})$$

When $y^{(i)} = 1$, i.e., actual positive

$$\ell(\theta) = -\ log(\hat{y}^{(i)})$$

If $\hat{y}^{(i)} = 1$, Loss = 0

If $\hat{y}^{(i)} = 0$, Loss = $+\infty$

When $y^{(i)} = 0$, i.e., actual negative

$$\ell(\theta) = -\ log(1 - \hat{y}^{(i)})$$

If $\hat{y}^{(i)} = 0$, Loss = 0

If $\hat{y}^{(i)} = 1$, Loss = $+\infty$

Logistic Regression

Finding θ

$$\ell(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}log(\hat{y}^{(i)}) + (1 - y^{(i)})log(1 - \hat{y}^{(i)})$$

Find partial derivative

To simplify, lets find the derivative for a **single** sample

Logistic Regression

Finding θ

$$\ell(\theta) = ylog(\hat{y}) + (1 - y)log(1 - \hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = \theta_0 + \theta_1 x$$

Want to find $\dfrac{\partial \ell}{\partial \theta}$

Using Chain Rule

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial \theta}$$

Logistic Regression

Finding θ

Logistic Regression

Finding θ

Summing over all samples

$$\frac{\partial \ell}{\partial \theta} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \cdot (\hat{y}^{(i)} - y^{(i)})$$

In matrix form

$$\nabla_{\theta}(\ell(\theta)) = \frac{1}{m} X^T (\hat{Y} - Y)$$

Logistic Regression

Summary

Model:

$$\hat{y} = \sigma(\theta_0 + \theta_1 x)$$

Loss:

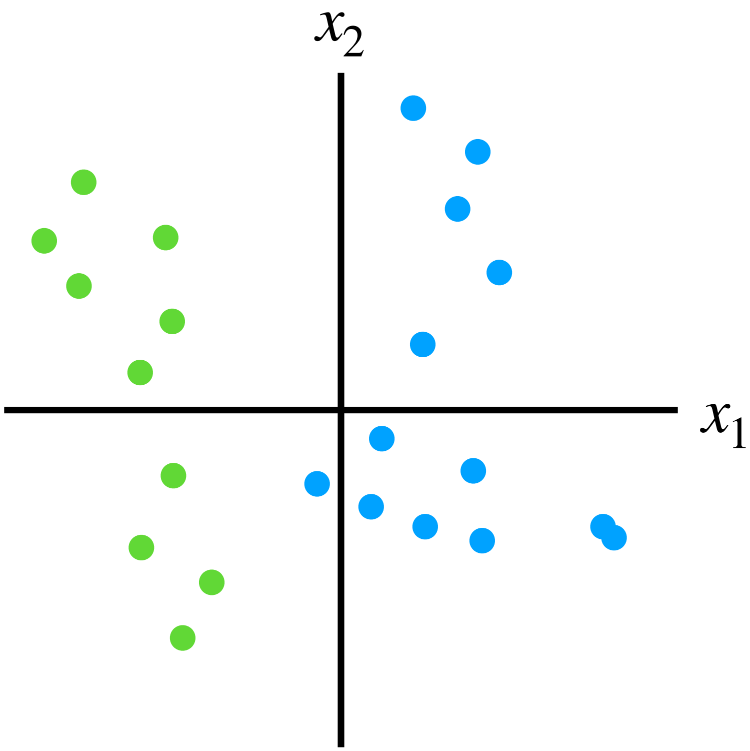
$$\ell(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)})$$

Gradient:

$$\nabla_\theta(\ell(\theta)) = \frac{1}{m} X^T(\hat{Y} - Y)$$

Logistic Regression

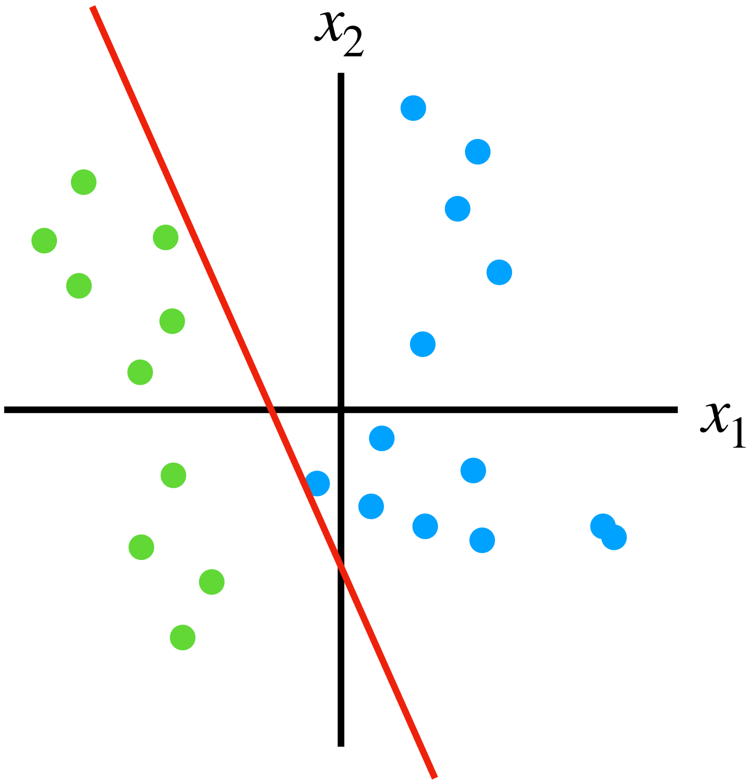
Summary



Logistic Regression

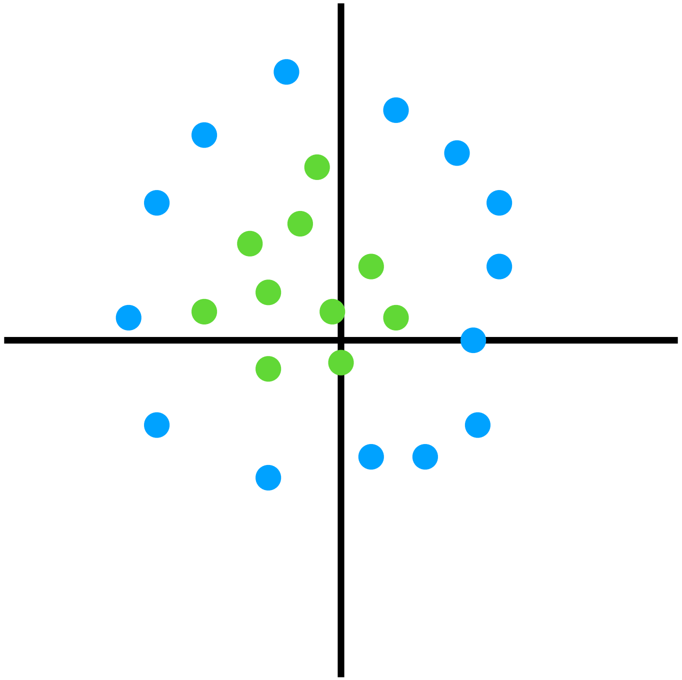
Summary

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$



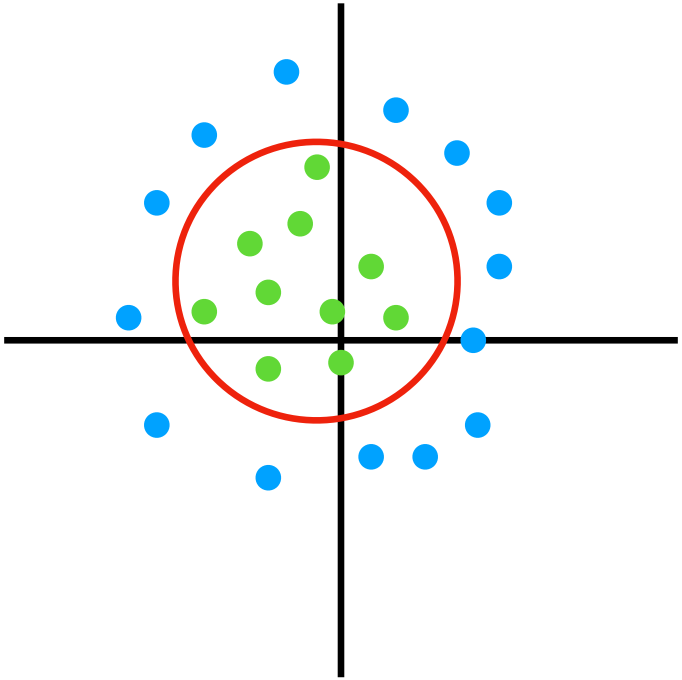
Logistic Regression

Summary



Logistic Regression

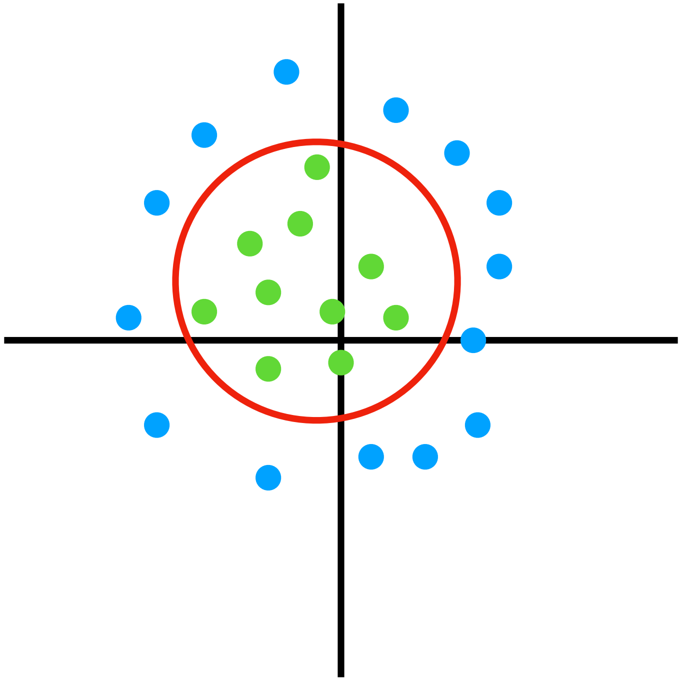
Summary



Logistic Regression

Summary

$$x_1^2 + x_2^2 = r^2$$

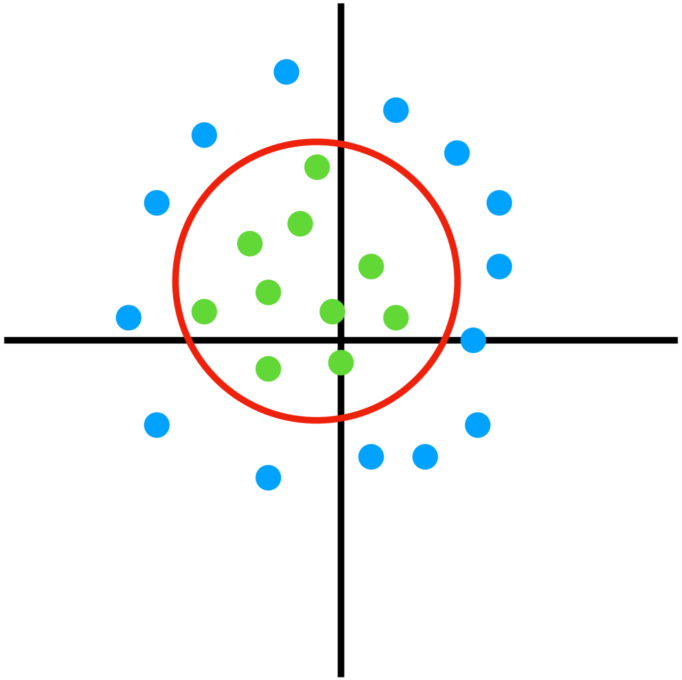


Logistic Regression

Summary

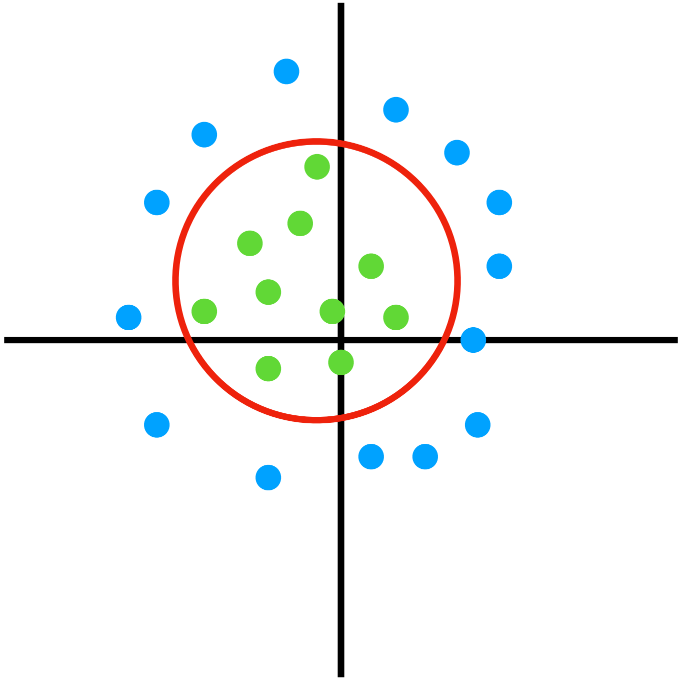
$$x_1^2 + x_2^2 = r^2$$

$$\theta_1^2(x_1^2 + x_2^2) = \theta_0^2$$



Logistic Regression

Summary



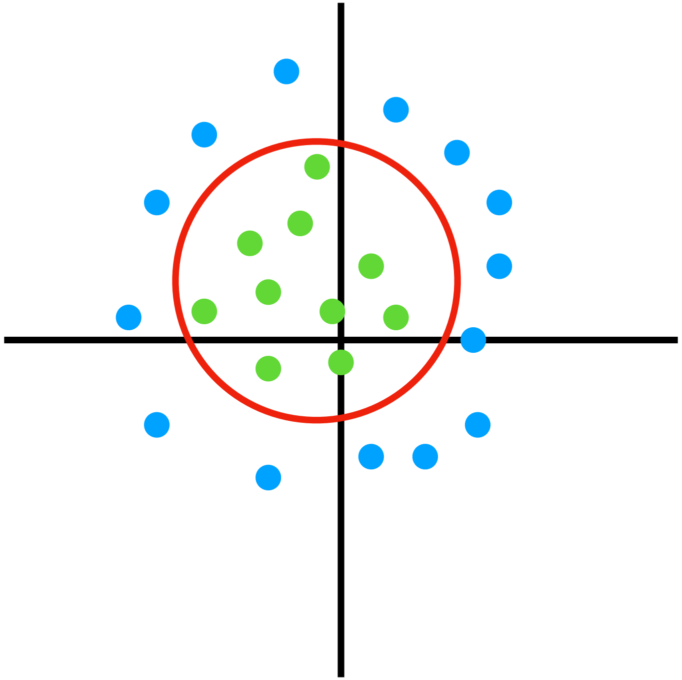
$$x_1^2 + x_2^2 = r^2$$

$$\theta_1^2(x_1^2 + x_2^2) = \theta_0^2$$

$$\sqrt{\theta_1^2(x_1^2 + x_2^2)} = \sqrt{\theta_0^2}$$

Logistic Regression

Summary



$$x_1^2 + x_2^2 = r^2$$

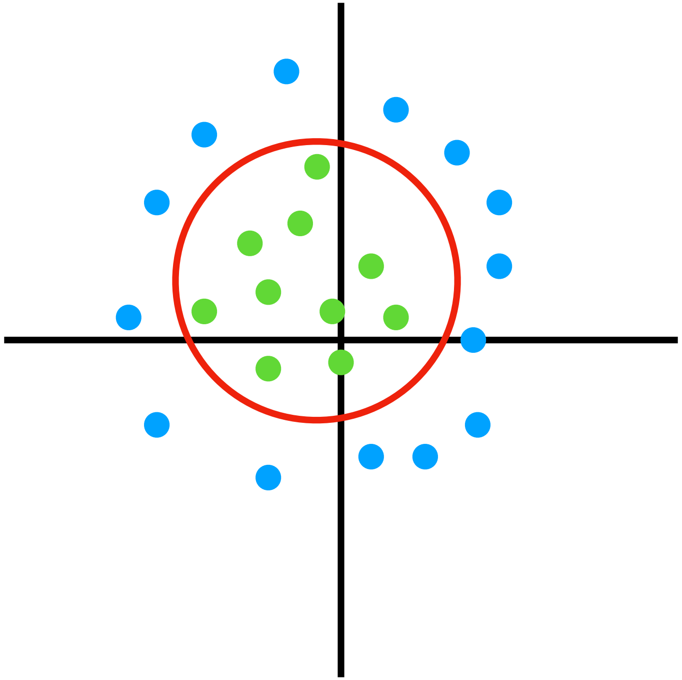
$$\theta_1^2(x_1^2 + x_2^2) = \theta_0^2$$

$$\sqrt{\theta_1^2(x_1^2 + x_2^2)} = \sqrt{\theta_0^2}$$

$$\theta_1\sqrt{(x_1^2 + x_2^2)} = \theta_0$$

Logistic Regression

Summary



$$x_1^2 + x_2^2 = r^2$$

$$\theta_1^2(x_1^2 + x_2^2) = \theta_0^2$$

$$\sqrt{\theta_1^2(x_1^2 + x_2^2)} = \sqrt{\theta_0^2}$$

$$\theta_1\sqrt{(x_1^2 + x_2^2)} = \theta_0$$

$$\hat{y} = \theta_1\sqrt{(x_1^2 + x_2^2)} - \theta_0$$

Next Class

- Homework Discussion
- More classification algorithms